# Introducción a los multiprocesadores

J. Daniel García Sánchez (coordinador)
David Expósito Singh
Javier García Blas
Óscar Pérez Alonso
J. Manuel Pérez Lobato

Arquitectura de Computadores
Grupo ARCOS
Departamento de Informática
Universidad Carlos III de Madrid

## 1. Estructura del módulo

Este módulo está estructurado en cuatro lecciones:

- **Memoria compartida simétrica**. Introduce el concepto de arquitectura multiprocesador y las alternativas de diseño en máquinas de memoria compartida. Presenta el problema de la coherencia de caché y las alternativas para su resolución. Así mismo ofrece detalles sobre los protocolos de espionaje (*snooping*).
- **Modelos de consistencia de memoria**. Introduce el concepto de consistencia de memoria y el modelo de consistencia secuencial. Presenta también modelos de consistencia más relajada, incluyendo el modelo de adquisición/liberación. Además presenta detalles del modelo de memoria de los procesadores Intel.
- **Sincronización**. Presenta el problema de la sincronización en máquinas de memoria compartida. También presenta las distintas alternativas de primitivas hardware sobre las que se puede implementar la sincronización. Introduce los conceptos de cerrojo y barrera así como sus alternativas de diseño.
- **Memoria compartida distribuida**. Profundiza en el concepto de memoria compartida distribuida y presenta los protocolos basados en directorio.

## 2. Memoria compartida simétrica

Esta lección tiene la siguiente estructura general:

1. Introducción a las arquitecturas multiprocesador.
2. Arquitecturas de memoria compartida centralizada.
3. Alternativas de coherencia de caché.

4. Protocolos de espionaje.

5. Rendimiento en SMPs.

## 2.1. Introducción a las arquitecturas multiprocesador

En los últimos años múltiples razones han contribuido a incrementar el interés por las máquinas multiprocesador. Por una parte, los beneficios cada vez menores del paralelismo a nivel de instrucción y el crecimiento en el consumo de energía hacen necesario mecanismo alternativos para mejorar el rendimiento de los computadores. Además, nuevos segmentos de mercado como los servicios basados en la nube y el software como servicio han situado cada vez mayores demandas sobre los servidores de alto rendimiento. Por último, el crecimiento de las aplicaciones intensivas en datos hacen necesario poder procesar cada vez mayores volúmenes de datos.

Para poder explotar el paralelismo ofrecido por las arquitecturas con varios procesadores es necesario explotar el paralelismo a nivel de hilo, que asume la existencia de varios contadores de programa. Si bien este modelo de programación se ha usado bastante en el dominio de la computación científica, su uso fuera de este dominio es relativamente reciente.

Un multiprocesador es un computador formado pro varios procesadores altamente acoplados con dos características importantes. Por una parte, la coordinación y uso de los procesadores está bajo el control de un único procesador. Por otra parte, toda la memoria es compartida con lo que existe un único espacio de direcciones global compartido por todos los procesadores.

Para hacer uso de un multiprocesador los modelos de programación más frecuentemente usados son el procesamiento paralelo, el procesamiento de peticiones y la multiprogramación. Desde un punto de vista físico las alternativas van desde los CMP (*chip multi-processors* o *multi-core*) hasta el uso de múltiples chips o el uso de multicomputadores.

Para poder realizar un aprovechamiento adecuado de un multiprocesador que tenga **n** procesadores se necesita descomponer el programa a ejecutar en **n** hilos de ejecución. Estos hilos de ejecución se pueden identificar de forma explícita por el programador, crearse por el sistema operativo a partir de las peticiones recibidas o generarse por el compilador. Es importante resaltar que, en cualquier caso, el número de instrucciones a ejecutar por cada hilo debe ser suficiente para compensar el coste de creación y destrucción de los hilos.

Desde el punto de vista de la memoria, un multiprocesador puede seguir dos diseños diferenciados. El uso de una memoria compartida centralizada da lugar los SMP (*symmetric multiprocessor*) que usan memoria UMA (*Uniform Memory Access*). El uso de memoria compartida distribuida da lugar a las máquinas DSM (*Distributed Shared Memory*) que usan memoria NUMA (*Non-Uniform Memory Access*).

## 2.2. Arquitecturas de memoria compartida centralizada

Una razón para el uso de memoria compartida centralizada es la existencia de memorias caché cada vez más grande en el procesador, lo que reduce la demanda efectiva sobre la memoria principal.

Dentro de la memoria caché se distinguen dos tipos de datos: datos privados y datos compartidos. Los problemas se plantean por la existencia de datos compartidos, ya que al poder estar el dato presente en más de una memoria caché se da la oportunidad de que las distintas copias puedan tener valores distintos. Este problema se conoce como el problema de la *coherencia* (o *incoherencia*) de caché.

Se dice que un sistema de memoria es *coherente* si cualquier lectura de una posición de memoria devuelve el valor que se ha escrito más recientemente en dicha posición de memoria.

Debe distinguirse entre dos problemas distintos, pero íntimamente relacionado. La *coherencia* determina qué valor se devuelve en el caso de una lectura. Por otra parte, la *consistencia* determina en

qué momento una escritura se hace visible al resto de procesadores. De esta forma, la coherencia define el comportamiento de las lecturas y escrituras sobre una única posición de memoria, mientras que la consistencia define el comportamiento de las lecturas y escrituras sobre una posición de memoria con respecto a los accesos a otras posiciones de memoria.

Para que se pueda garantizar la consistencia deben cumplirse tres condiciones: la preservación del orden de programa, la existencia de una vista coherente de la memoria y la serialización de las escrituras.

## 2.3. Alternativas de coherencia de caché

Un procesador coherente debe ofrecer dos propiedades fundamentales para el rendimiento. Por una parte, la migración de datos compartidos y por otra parte la replicación de datos compartidos leídos. Estas propiedades se pueden integrar en un protocolo de coherencia de caché.

Existen dos familias de protocolos de coherencia de caché: protocolos basados en directorio y protocolos de espionaje (o *snooping*). En los protocolos basados en directorio el estado de compartición se mantiene en una estructura de datos (el *directorio*), que puede ser centralizado o distribuido. En los protocolos de espionaje cada caché mantiene la información del estado de compartición de cada bloque que tiene.

## 2.4. Protocolos de espionaje

En un protocolo de espionaje se pueden usar dos estrategias la invalidación de escrituras y la actualización de escrituras. En la invalidación se garantiza que un procesador tiene acceso exclusivo a un bloque antes de realizar una escritura sobre el mismo. Previamente, deben *invalidarse* el resto de copias. Por el contrario, en la difusión de escrituras cada vez que se realiza una escritura sobre una posición de memoria, esta debe difundirse a todas las cachés, que, eventualmente modificarían su copia del bloque. Como esta estrategia consume más ancho de banda, se suele optar por la estrategia de invalidación.

En la invalidación, cuando un procesador necesita invalidar un dato debe adquirir el bus de memoria y difundir la dirección a invalidar. Todos los procesadores estarán observando (*espiando*) el bus para comprobar si la tienen una copia del bloque invalidado. De esta manera el bus se utiliza como mecanismo de serialización impidiendo que se produzcan escrituras simultáneas. En el caso en que se produzca un fallo de caché, el comportamiento varía si la estrategia de escritura es de escritura inmediata o de post-escritura. Con escritura inmediata, la memoria tiene siempre la última escritura inmediata y el dato debe traerse de allí. Sin embargo, en el caso de post-escritura, si un procesador tiene una copia modificada del bloque contestará al fallo de caché del otro procesador.

El protocolo más básico basado en espionaje e invalidación es el protocolo MSI. Este protocolo usa una máquina de tres estados para cada bloque de la memoria caché. Los cambios de estado pueden generarse peticiones del procesador o peticiones del bus. Los tres estados básicos se definen en términos del estado del bloque. El bloque puede haber sido modificado (M), estar compartido (S) o haber sido invalidado (I).

Existen extensiones sobre este protocolo básico. El protocolo MESI añade un cuarto estado exclusivo que el bloque reside en una única memoria caché pero no está modificado. Otras alternativas son el protocolo MESIF (usado en Intel Core i7) y el protocolo (MOESI) usado en el AMD Opteron.

## 2.5. Rendimiento en SMPs

El uso de políticas de coherencia de caché tiene impacto sobre la tasa de fallos y, por tanto, sobre el rendimiento. Aparecen dos tipos de fallos. Los fallos de compartición verdadera y los fallos de falsa

compartición. Estos últimos se deben a accesos por distintos procesadores a palabras que pertenecen a un mismo bloque y sus invalidaciones correspondientes.

# 3. Modelos de consistencia de memoria

Esta lección tiene la siguiente estructura general:

1. Modelo de memoria.
2. Consistencia secuencial.
3. Otros modelos de consistencia.
4. Caso de uso: Intel.

## 3.1. Modelo de memoria

Un modelo de consistencia de memoria define el conjunto de reglas que definen como se procesan, por parte del sistema de memoria, operaciones de lectura y escritura que provienen de múltiples procesadores. Por tanto, dicho modelo puede verse como un contrato entre el programador y el sistema.

En general, cualquier modelo de consistencia de memoria determina la validez de las posibles optimizaciones que pueden realizarse sobre programas correctos. Así, el modelo de memoria define la interfaz entre un programa y sus distintos transformadores (como el compilador o el propio hardware sobre el que se ejecuta el programa).

En el caso de un sistema con un único procesador las operaciones de memoria ocurren en orden de programa. Esto es, la semántica viene definida por el orden de programa secuencial

## 3.2. Consistencia secuencial

En el caso de multiprocesadores, el modelo de memoria sobre el que es más simple razonar es el modelo de consistencia secuencial. El concepto de consistencia secuencial fue definido por Leslie Lamport en 1979.

La consistencia secuencial impone dos restricciones: orden de programa y atomicidad. Por una parte, las operaciones de memoria de un programa deben hacerse visibles a todos los procesos en el *orden de programa*. Por otra parte todas las operaciones de memoria deben ser *atómicas* requiriendo que nada que un procesador haga después de que haya visto el nuevo valor de una escritura se hace visible a otros procesos antes de que hayan visto el valor de esa escritura.

La consistencia secuencial restringe el orden de las operaciones de memoria siendo un modelo simple para razonar sobre programas paralelos, pero incluso reordenaciones simples que son válidas en un contexto monoprocesador, dejan de ser válidas.

Existe un conjunto de condiciones suficiente que garantizan la existencia de la consistencia secuencial. No obstante, puede haber condiciones menos exigentes.

## 3.3. Otros modelos de consistencia

Existen otros modelos de consistencia menos exigentes que el modelo de consistencia secuencial. Estos modelos pueden definirse en términos de qué operaciones pueden adelantar a otras.

En el ordenamiento débil las operaciones realizadas sobre memoria se dividen en dos tipos: operaciones de datos y operaciones de sincronización. Las operaciones de sincronización actúan como barreras. De esta manera, todas las operaciones de sincronización deben ejecutarse en orden de programa y

para las operaciones de datos se admiten reordenaciones siempre que éstas no traspasen una barrera impuesta por una operación de sincronización.

Otro modelo más relajado que el ordenamiento débil y la consistencia de adquisición/liberación. En este caso, las operaciones de sincronización pueden ser de dos tipos: adquisición y liberación. Una adquisición debe completarse antes que todos los accesos de memoria subsiguientes. En el caso de la liberación, todos los accesos a memoria previos a la misma deben completarse antes de iniciar dicha liberación.

### 3.4. Caso de uso: Intel

En el caso de la familia de procesadores de Intel, el modelo de memoria se ha ido definiendo para las diversas generaciones de procesadores.

Por una parte, el modelo define en qué casos las operaciones de memoria pueden considerarse atómicas. También define las diferentes condiciones para que una operación pueda establecer bloque del bus de memoria.

Por otra parte, el modelo define también instrucciones de sincronización (instrucciones de barrera).

Finalmente, el modelo define las reglas del modelo de memoria dentro del procesador, así como las reglas del modelo de memoria para accesos entre múltiples procesadores.

## 4. Sincronización

Esta lección tiene la siguiente estructura general:

1. Introducción.
2. Primitivas hardware.
3. Cerrojos.
4. Barreras.

### 4.1. Introducción

En sistemas de memoria compartida la comunicación se realiza a través de escrituras y lecturas sobre dicha memoria compartida. Para evitar problemas en el acceso concurrente a la memoria se hace necesario establecer mecanismos de sincronización en el acceso a las variables compartidas. Se distinguen dos casos la comunicación 1-1 (comunicación entre dos procesos) y la comunicación colectiva (comunicación entre un número arbitrario de procesos).

En la comunicación 1-1 se debe asegurar que la recepción se produce después que la emisión. De forma general se debe garantizar que los accesos a memoria se producen garantizando la *exclusión mutua*. Es decir, que solamente uno de los procesos puede acceder a la vez a la variable compartida.

En la comunicación colectiva se hace necesario coordinar múltiples accesos a variables garantizando que las escrituras se realizan sin interferencias y que las lecturas esperan a que los datos estén disponibles. Otra vez, se hace necesario que los accesos a las variables compartidas se realizan en exclusión mutua. En este caso, se debe garantizar que no se lee un resultado hasta que todos los procesos han ejecutado su sección crítica.

### 4.2. Primitivas hardware

El modelo de consistencia puede ser insuficiente y complejo por lo que se suele complementar con primitivas de lectura-modificación-escritura. Estas pueden ser de distinto tipo como:

- Instrucción test-and-set.
- Instrucciones de intercambio.
- Instrucciones de captación y operación.
- Instrucciones de comparación e intercambio.
- Instrucciones de almacenamiento condicional.

### 4.3. Cerrojos

Un cerrojo es un mecanismo que asegura la exclusión mutua. Tiene dos operaciones asociadas: la operación de adquisición (*lock*) y la de liberación (*unlock*).

La operación de adquisición del cerrojo puede implicar una espera. Existen dos alternativas para implementar el mecanismo de espera: la espera activa y el bloqueo. Con la espera activa, el proceso queda en un bucle que constantemente consulta el valor de la variable de control de la espera. Esta es la idea básica que subyace en un *spin-lock*. Con el bloqueo, el proceso queda suspendido y se cede el procesador a otro proceso. Por tanto, el bloqueo requiere la participación de un planificador de procesos, que típicamente forma parte del sistema operativo o de un sistema de soporte en tiempo de ejecución (*run-time*). En la selección de la alternativa es necesario tener en cuenta los compromisos de coste de tiempo de ejecución.

En general, en el diseño de un mecanismo de cerrojos intervienen tres elementos de diseño: el método de adquisición, el método de espera y el método de liberación.

La implementación más simple de un cerrojo se basa en el uso de una variable compartida capaz de tomar dos valores (abierto o cerrado). Para reducir los accesos a memoria del mecanismo de espera, una técnica que se puede usar es la del retardo exponencial. Otra optimización, que se puede usar en algunos casos es utilizar la misma variable para sincronizar y comunicar.

Por último, otro problema de las implementaciones más simples es que no fijan el orden de adquisición del cerrojo lo que puede dar lugar a problemas de inanición. La solución a este problema es hacer que el cerrojo se adquiera por antigüedad en la solicitud garantizando el orden FIFO. Para ello se pueden usar cerrojos con etiquetas o bien cerrojos basados en colas.

### 4.4. Barreras

Una barrera permite sincronizar varios procesos en algún punto y garantiza que ningún proceso supera la barrera hasta que todos han llegado. Así las barreras se usan para sincronizar las fases de un programa. La implementación más simple de una barrera es la de una barrera centralizada donde se asocia a la barrera un contador de los procesos que han llegado a la misma. Para evitar los problemas de reutilización de una barrera en un bucle se usan barreras con inversión de sentido. Por último, para evitar los problemas de escalabilidad y contención en el acceso a variables compartidas se pueden usar barreras jerárquicas donde se estructuran los procesos de llegada y liberación en forma de árbol.

## 5. Memoria compartida distribuida

Esta lección tiene la siguiente estructura general:

1. Introducción a memoria compartida distribuida.
2. Bases del protocolo de directorio.
3. Protocolo basado en directorio.

## 5.1. Introducción a memoria compartida distribuida

Los protocolos de espionaje requieren comunicación con todas las cachés tanto en cada fallo de caché como en cada escritura de dato compartido. No obstante, su uso se debe a la ausencia de una estructura de datos centralizada lo que ocasiona un bajo coste de implementación. Cuando el número de procesadores crece, los problemas de escalabilidad del protocolo de espionaje crecen.

En las máquinas de memoria compartida distribuida (DSM) el tráfico de coherencia en el bus puede suponer un cuello de botella. La otra alternativa es el uso de protocolos basados en directorio que mantienen el estado de compartición. Estos se pueden usar a dos niveles. En procesadores SMP se puede usar un directorio centralizado en la memoria un en la caché de último nivel. En máquinas DSM se puede usar un directorio distribuido que evita los cuellos de botella derivados del tráfico en el bus.

En general, la idea básica de un protocolo basado en directorio es mantener información sobre el estado de cada bloque de caché. Esta información incluye una lista de las cachés que tienen una copia del bloque, así como los bits de estado del bloque.

Dentro de un procesador multicore, esta información se puede representar usando un bit para cada core. Así solamente se envían invalidaciones a las cachés marcadas en este mapa de bits, evitando los mensajes de multidifusión (*broadcast*). Si bien se evitan estos mensajes cuando el número de procesadores crece, el directorio centralizado plantea problemas de escalabilidad.

El directorio distribuido resuelve estos problemas de escalabilidad distribuyendo el directorio con la memoria. Así, cada directorio tiene información sobre los bloques de la memoria local asociada.

## 5.2. Bases del protocolo de directorio

Un protocolo de directorio mantiene el estado de cada bloque que puede ser: compartido (uno o más nodos tienen el bloque en caché y el valor en memoria está actualizado), no cacheado (ningún nodo tiene una copia del bloque) o modificado (solamente un nodo tiene una copia del bloque en caché y lo ha escrito). En particular el protocolo debe tratar tanto los fallos de lectura como las escrituras en bloques compartidos limpios.

## 5.3. Protocolo basado en directorio

En chips multi-core la coherencia interna se mantiene mediante un directorio centralizado. Dicho directorio puede actuar como directorio local en el caso de memoria compartida distribuida. En este caso la implementación del protocolo requiere el tratamiento de de las transiciones locales y distribuidas.